Статический анализ последовательных программ в системе автоматизированного распараллеливания САПФОР*

Н.А. Катаев

Московский Государственный Университет имени М.В. Ломоносова, Институт Прикладной Математики имени М.В. Келдыша РАН

Для использования последовательных языков программирования при создании параллельной программы необходимо наличие эффективных средств анализа. В статье описывается подсистема анализа, входящая в САПФОР. Разработка подсистемы осуществляется в три этапа. В статье представлены результатырешения задачи анализа скалярных приватных переменных. Приводится описание структуры подсистемы анализа, представлены разработанные алгоритмы анализа, используемые в подсистеме анализа.

1. Введение

Развитие суперкомпьютерной отрасли идет очень быстрыми темпами. Эффективное использование все возрастающих мощностей наталкивается на значительные трудности. В японском суперкомпьютереKComputer, занимающем первое место в списке Top500[1] и показавшем на тесте Linpackпроизводительность свыше 10 ПФлопс, используется 705 024 вычислительных ядер. Разрабатывать и отлаживать прикладное программное обеспечение, максимально использующее ресурсы систем такого уровня, очень сложно.

За последнее время к широко распространённым и давно развивающимся технологиям параллельного программирования (MPI, SHMEM, DVM,OpenMP) добавились низкоуровневые технологиидля гетерогенных систем (CUDA, OpenCL). Использование этих технологий требует четкого понимания“информационной структуры” [2] разрабатываемой программы, то есть связи различных элементов программы между собой. Такими элементами чаще всего являются операции. В совокупности со связями между ними операции образуют различные графовые модели программы. Такие модели оказываются достаточно сложными, и изучение их без специализированных средств тяжелая задача для разработчиков.

Часто параллельные программы создаются на основе ранее существовавших последовательных программ.В этом случае сложность изучения информационной структуры может усугубиться отсутствием разработчиков исходных кодов. Трудности возникают, когда задача по распараллеливанию решается сторонней организацией.

В таких случаях становятся незаменимы автоматизированные средства создания параллельных программ. Такие системы обращаются за помощь к программистам лишь при крайней необходимости. Неотъемлемой частью таких систем являются подсистемы анализа программ.

Анализ и преобразование программ в распараллеливающих системах может выполняться как в полностью автоматическом режиме, так и в режиме диалога с пользователем.

К полностью автоматическим средствам относятся автоматически распараллеливающие компиляторы, поставляемые компаниями Intel, Microsoft, SunMicrosystems, IBMи др. Другим примером является автоматический распараллеливатель, реализованный компанией Оптимизирующие технологии [3] в рамках компилятора GCC. В данном случае распараллеливание производится с использованием технологии OpenMPи Универсальной Библиотеки Трансляции (УБТ)[3]в рамках которой реализованы алгоритмы анализа и оптимизации.

К диалоговым распараллеливающим системам относятся такие комплексы, как ParaWise[4], ДВОР [5], САПФОР[6]. Подсистема анализа программ, разрабатываемая в рамках данного исследования, входит в состав САПФОР.

*Работа поддержана Программами президиума РАН №14, №15 и №17, грантом РФФИ № 10-07-00211.

2. САПФОР

Система Автоматизированного Распараллеливания Фортран Программ (САПФОР) разрабатывается в Институте Прикладной Математики им. М.В. Келдыша РАН. Входным языком системы является Fortran, а результат распараллеливания представляет собой программу на языкеFortranOpenMP, FortranDVM/OpenMPили FortranDVMH[7, 8, 9].Все эти языки являются расширением стандартного языка Fortran 95 директивами параллелизма. Высокий уровень выходного языка позволяет продемонстрировать пользователю результат распараллеливания в понятных для него терминах, кроме того директивы распараллеливания могут быть проигнорированы компиляторами, не поддерживающими соответствующий язык. Для данных языков существуют развитые средства отладки функциональности и эффективности.

Отличительной особенностью САПФОР является использование автоматически распараллеливающего компилятора[10], преобразующего потенциальнопараллельную программу в ее параллельную версию для заданной ЭВМ. При этом предварительный анализ программы может выполняться в полуавтоматическом режиме. Для уточнения свойств последовательной программы, выявленных анализатором, используются либо диалоговая оболочка [11], либо специальные аннотации в тексте программы. В САПФОР наиболее сложный этап распараллеливания (получение параллельной версии программы) выполняется полностью автоматически.

Основными компонентами САПФОР являются анализаторы последовательных программ, блоки преобразования последовательных программ в параллельные программы (эксперты), диалоговая оболочка для взаимодействия с пользователем, генератор кода, создающий на основе принятых экспертом решений параллельную версию программы. Связь между всеми компонентами осуществляется через базу данных, хранящую всю необходимую информацию для построения параллельной версии исходной программы.Схема САПФОР показана на **Рис. 1**.

Рис. 1. Схема САПФОР

3. Подсистемы анализа

В САПФОР может одновременно использоваться несколько подсистем анализа (анализаторов), в этом случае запуск каждого следующего анализатора уточняет результаты уже проведенного анализа.Подсистемы анализа выполняют две основные функции.

Во-первых, структура программы должна быть сохранена в соответствии с требованиями базы данных САПФОР. Из базы данных должна быть доступна информация о программных единицах (далее “ПЕ”), образующих программу и принадлежности ПЕ к конкретным файлам.Для каждой ПЕ сохраняется дерево циклов с информацией о тесной вложенности отдельных циклов. Должен быть сохранен граф управления программы и информация о COMMON-блоках (для Fortran-программ).

Во-вторых, в результате анализа должны быть выявлены зависимости по данным [12], содержащиеся в программе.

Для всех циклов программы предоставляется информация следующего вида:

- информационные зависимости между витками цикла (выходные (output), прямые (flow), обратные (anti));
- редукционные зависимости между витками цикла;
- приватные переменные (скаляры и массивы);
- регулярные зависимости по массивам. Наличие регулярной зависимости означает, чтомежду витками имеет место прямаязависимость, и виток зависит от некоторого количества (не более константы) соседних предыдущих витков. Такой цикл допускает конвейерное выполнение.

Выделяется два типа анализаторов: динамические и статические.

Динамические анализаторы выполняют анализ программы в процессе ее выполнения на представительных данных.Динамический анализатор позволяет:

- точнее оценить количество витков циклов и время их выполнения, а также размеры динамических массивов;
- обнаруживать приватные переменные и зависимости (информационные и регулярные) в случаях косвенной индексации и с учетом вводимых данных.

Основным недостатком является то, что динамический анализне способен обеспечить эксперта информацией для всевозможных наборов входных данных.

Статические анализаторы работают по тексту программы.

В САПФОР существует 4 статических анализатора, способных обнаруживать различные типы свойств исследуемых программ. Три из них выполняют анализ на основе исходного текста программы, а входом для четвертого служит база данных, подготовленная одним из трех анализаторов. После выполнения анализа любым из 4 анализаторов может быть запущен эксперт для получения параллельной версии программы, при этом эффективность распараллеливания будет зависеть от полноты выполненного анализа.

Первый анализатор основывается на Sage++[13]представлении программ. Основной его задачей является сохранение в базе данных структуры программы и выявление редукционных зависимостей без осуществления межпроцедурного анализа.

Следующий анализатор[14] является развитием первого, но применяется только для программ, написанных на Fortran 77. Данный анализатор способен обнаруживать информационные зависимости.В ряде случаев используемый в анализаторе алгоритм может быть применен для межпроцедурного анализа. Кроме того анализатор способен определять условия возникновения зависимостей в программе.

Еще один анализатор [15] в качестве основного средства анализа использует Универсальную Библиотеку Трансляции (УБТ) [3]. Анализ программ выполняется на основе внутреннего представления GIMPLEкомпилятора GCC.Единство данного представления для различных языков программирования позволяет создать единую систему анализа программ, написанных на языках Fortran и C/C++. В случае языка C++ использование GCCявляется единственным возможным решением, так как компоненты системы не рассчитаны на сложные конструкции объектно-ориентированного языка.Проблема может быть решена с помощью распараллеливания исходной программы непосредственно на уровне GIMPLE.Но поддержание данного анали-

затора в актуальном состоянии наталкивается на многочисленные трудности [15], кроме того УБТявляется коммерческим продуктом, что усложняет возможность ее использования.

В статье будет рассмотрен последний из анализаторов. В отличие от остальных он работает по информации, содержащейся в базе данных САПФОР. Вследствие этого, он потенциально не зависит от языка, на котором написана анализируемая программа. Для выполнения анализа программ, написанных на языках отличных от Fortran, достаточно создать средство сохраняющее структуру программы в терминах базы данных САПФОР. Разработку данного анализатора можно разбить на три этапа:

1. Создание подсистемы, выполняющей анализ приватных скалярных переменных с выполнением межпроцедурного анализа и учетом псевдонимов, появляющихся вследствие использования COMMON-блоков и модулей. Уточнение анализа редукционных переменных.
2. Интеграция второго из рассмотренных анализаторов[14] в разрабатываемыйанализатор, с целью определения информационных и регулярных зависимостей.
3. Создание подсистемы, выполняющейанализ приватных массивов.

4. Структура анализатора

Анализатор образуют три части:

- классы, поддерживающие внутреннее представление;
- загрузчик (frontend),осуществляющий построение внутреннего представления анализатора на основе входных данных (например, базы данных САПФОР);
- модули анализа, выполняющие различные виды анализа над внутренним представлением анализатора.

Результатом работы анализатора является обновленная база данных САПФОР. Обновление базы данных выполняется сразу в процессе анализа, чтобы снизить объем используемой анализатором памяти.

Внутреннее представлениене зависит от языка входной программы и моделирует ее информационную структуру с помощью следующих графовых моделей:

- граф оператора или выражения;
- граф управления процедуры (подпрограммы или функции);
- дерево циклов процедуры;
- граф вызовов программы;
- граф используемой в программе памяти.

Граф оператора – ориентированный граф, вершинами которого являются операции, а дуги показывают направление потока информации между операциями. Выделяется множество входных вершин, которые описывают используемые в операторе данные (переменные или константы), и множество выходных вершин, которые описывают результат применения оператора.Пример графа операториз строки 8 программы на **Рис. 2** приведен на **Рис. 3**.

Граф управления процедуры – ориентированный граф, вершинами которого являются базовые блоки [12], а дуги показывают направлениепотока управления в программе. Базовые блоки содержат операторы процедуры, каждый из которых представлен графом оператора. Выделяются начальный и конечный базовый блок, которые не содержат исполняемых операторов программы. Данные вспомогательные блоки используются модулями анализа.

Граф вызовов программы содержит информацию о точках вызовов процедур.

Граф используемой в программе памяти – ориентированные граф, вершинами которого являются участки памяти (ПЕ, COMMON-блоки, локальные переменныеи др.), а дуги показывают пересечение между участками памяти в программе. Для пересечений по памяти содержится информация об их границах. Если скалярная переменная ссылается на элемент массива, то номер данного элемент доступен из графа памяти.

Используемые в программе именованные константы в граф памяти не попадают. Если COMMON-блок или модуль явно не указан в процедуре, но используется косвенно (например, при вызовах других процедур), данная информация также отображается в графе памяти. Пример графа памяти для программы на **Рис. 2** приведен на **Рис. 4**.

```
1. program Example
2. parameter( n = 100)
3. external add
4. integeri, j, add
5.
6. i = n
7. dowhile ( i< n * n)
8.           j = add( i, n)
9. enddo
10.
11.    call print( j)
12.    end
13.
14.    subroutine print( j)
15.    integer j, k
16.    common /a/ k
17.
18.    print *, "Two last elements: ", k, j
19.    end
20.
21.    integerfunction add( x, y)
22.    integer x, y, z
23.    common /a/ z
24.
25.         add = x + y
26.         z = x
27.         x = add
28.    end
```

Рис. 2. Программа, печатающая два последних элемента арифметической прогрессии

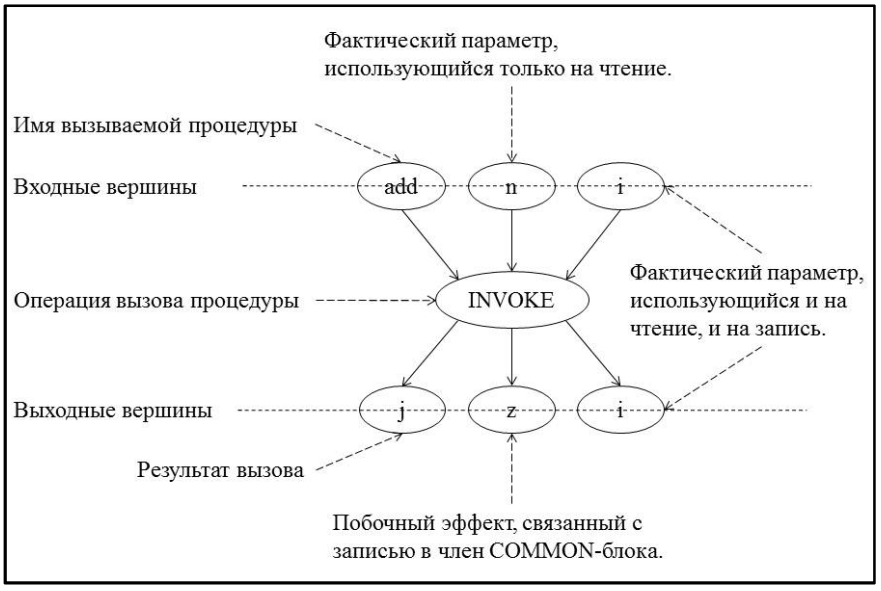

Рис.3.Граф оператора `j = add( i, n)`

В процессе анализа внутреннее представление модифицируется модулями анализа, с целью наиболее точного отражения свойств анализируемой программы. Модулями анализа выполняется следующая последовательность действий:

1. Упорядочивание процедур: построение последовательности обратного обхода глубинного остовного дерева графа вызовов [12]. Данное упорядочивание процедур используется на последующих шагах анализа.
2. Уточнение описания участков памяти, используемых в процедурах.
3. Определение возможного пересечения по памяти между формальными параметрами процедур и членами COMMON-блоков и модулей.

4. Определение приватных переменных.
5. Уточнение редукционных переменных необходимо, так как в базе данных, подаваемой на вход анализатору, сохранена информация о редукциях в программе, полученная без межпроцедурного анализа.

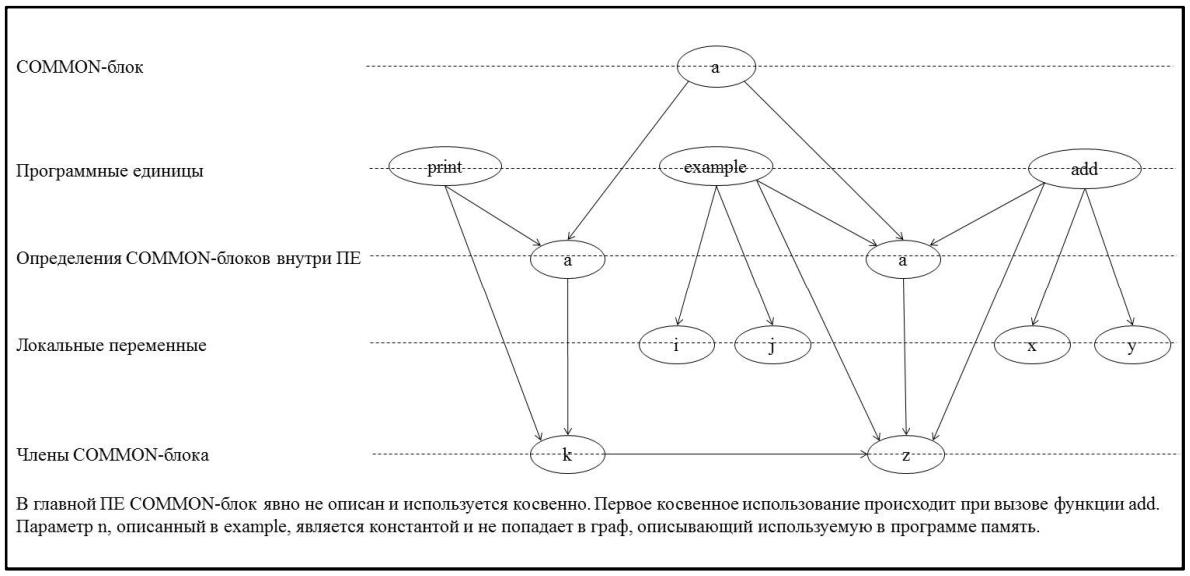

Рис. 4. Граф, используемой в программе памяти

Без использования межпроцедурного анализа невозможно определить, к каким участкам памяти будет происходить обращение при вызове некоторой процедуры. Из базы данных, подаваемой на вход анализатора, в качестве описания вызова доступен только список фактических параметров, при этом не известно: используются они на запись или на чтение. Кроме того, вызов процедуры может иметь побочный эффект, являющийся следствием использования глобальных данных (COMMON-блоков и модулей).

Анализ выполняется над вызываемой процедурой один раз, и полученные результаты используются для уточнения описания участков памяти, используемых во всех вызовах. Анализ выполняется в порядке определенном на первом шаге. При отсутствии рекурсивных вызовов процедур в программе это гарантирует, что при уточнении памяти, используемой в анализируемой процедуре, все вызываемые из нее процедуры уже проанализированы. Результаты выполненного анализа отображаются в графе оператора, содержащего вызов.

Результат выполнения данного шага виден на **Рис. 3**.В процессе анализа было установлено, что при вызове функции `add`в операторе из строки 8 программы на **Рис. 2**, параметр `n`, используется только на чтение, параметр`i`используется на чтение и на запись.Кроме того было установлено, что оператор использует на запись член COMMON-блока, явно не описанного в вызывающей процедуре.Данная информация повлекла за собой модификацию графа памяти. Обновленный граф памяти изображен на **Рис. 4**.

При некоторых вызовах процедур фактические параметры могут пересекаться по памяти. Это порождает соответствующую связь между формальными параметрами, которая должна быть учтена при последующем анализе. Фактические параметры могут быть также связаны с элементами COMMON-блоков или модулей, используемых в вызываемой процедуре.

Данная информация определяется на основе возможных значения формальных параметров. Под значениями понимаются те участки памяти из других ПЕ или COMMON-блоков, которым могут соответствовать формальные параметры ПЕ при некотором потенциально возможном вызове. Найденные связи добавляются в граф памяти программы.

Анализ выполняется в порядке противоположном порядку, определенному на первом шаге. Это гарантирует, что при анализе любой процедуры все вызывающие ее процедурыуже проанализированы, то есть известны все возможные значения формальных параметров вызывающих процедур. Это необходимо, в случае, когда формальные параметры вызывающей процедуры являются фактическими для вызываемой процедуры.

5. Анализ приватных переменных

Пусть Var_P – множество переменных некоторой программы P, MU_P – память, используемая в программе,$Operation_P$ – множество операций программы, $Loop_P$ – множество циклов программы.

Каждой переменной $v \in Var_P$ соответствует некоторый участок памяти $mu(v) \in MU_P$.При этом могут существовать переменные v_1,$v_2 \in Var$такие, что$mu(v_1) \cap mu(v_2) \neq \emptyset$.Такие переменные являются псевдонимами. К ним относятся указатели в Си, члены COMMON-блоков, эквивалентности, члены модулей в Fortran. Псевдонимы могут возникнуть при передаче параметров по ссылке.

Будем говорить, что операция $S_2 \in Operation_P$достижима изоперации $S_1 \in Operation_P$, если поток управления после прохождения операции S_1 достигает операции S_2. Отношение достижимости выполняется и в том случае, если операции S_1 и S_2 совпадают.

Рассмотрим множество итераций $Iteration_L$некоторого цикла$L \in Loop_P$.Введем на данном множестве отношение порядка в соответствии с порядком выполнения итераций в последовательной программе. Пронумеруем все итерации из данного множества (нумерация начинается с 0) и, говоря о некоторой итерации, будет иметь в виду ее номер. Тогда операцию $S \in Operation_P$, выполняющуюся в цикле $L \in Loop_P$,можно рассматривать, как множество выполнений заданной операции на различных итерациях цикла: $S = \{S^I : S^I \in I \wedge I \in Iteration_L\}$.

Переменная$v \in Var_P$ является приватной(private) для цикла $L \in Loop_P$, если:

1. Между любыми операциями$S_1^{I_1}, S_2^{I_2} \in Operation_P$, выполняющимися на разных итерациях цикла$L(I_1, I_2 \in Iteration_L \wedge I_1 \neq I_2)$, использующими память $mu(v)$ и такими, что $S_2^{I_2}$достижима из $S_1^{I_1}$, существует операция $S^{I_2} \in I_2$, использующая $mu(v)$ на запись, такая что S^{I_2} достижима из $S_1^{I_1}$, а $S_2^{I_2}$ достижима из S^{I_2}.
2. Между любыми операциями $S_1, S_2 \in Operation_P$, использующими память $mu(v)$, такими, что $S_1 \in L \wedge S_2 \notin L$, и$S_2$,достижима из$S_1$,существует операция $S \notin L$, использующая $mu(v)$ на запись.

Если выполняется условие (1) и все операцииS_2, для которых условие (2) не выполняется, читают одно и то же значение, записанное в участок памяти$mu(v)$ на последней итерации цикла, то переменная vявляется приватной по выходу (lastprivate).

Если выполняется условие (2) и все операции $S_2^{I_2}$, для которых условие (1) не выполняется, используют значение, записанное в участок памяти$mu(v)$, перед входом в цикл, то переменная vявляется приватной по входу (firstprivate).

Задачей анализа приватных переменных является обнаружение приватных переменных, переменных приватных по входу и по выходу для каждого естественногоцикла программы. Естественный цикл[12] имеет единственный входной узел, называемый заголовком цикла. Анализ выполняется над каждой из процедур, при этом процедуры рассматриваются в порядке определенном на шаге 1 последовательности действий,выполняемых модулями анализа. При этом используются результаты шагов 2 и 3.

На первом этапе разработки анализатора решалась задача анализа скалярных приватных переменных. Так как анализ условий, по которым происходят ветвления в программе, не производится (данная информация не содержится в базе данных САПФОР), анализ приватных по входу переменных не возможен.Необходимым условием приватности скалярной переменной по входу, являетсятребование того, чтобы использование данной переменной на чтение регулировалось некоторой конструкцией ветвления, расположенной в цикле.

Рассмотрим последовательность процедур $P_1^n = \{P_1, P_2, \dots, P_n\}$, такую что $\forall i = \overline{1, n-1}$существует вызов процедуры P_{i+1}из процедуры P_iи такую что $mu(P_1) \cap mu(P_2) \cap \dots \cap mu(P_n) \neq \emptyset$, где$mu(P_i)$ – участок памяти, используемой в процедуре P_i. Непустое пересечение возможно за счет передачи параметров по ссылке, обращения к членам COMMON-блоков и модулей.

Рассмотрим переменную $v \in Var_P$, описанную в P_nи такую, чтоучасток памяти$mu(v)$используется всеми процедурами последовательности P_1^n. В этом случае для решения задачи анализа приватных переменных не достаточно проанализировать только процедуру P_n.Операция

S_2 из условия (2) приватности переменной, использующая память$mu(v)$ может находиться в одной из процедур последовательности P_1^n, отличной от процедуры P_n.

Необходимо выполнить межпроцедурный анализ для всех последовательностей P_1^n, удовлетворяющих условию описанному выше. При таком анализе процедуры нужно рассматривать в направлении обратном направлению вызовов, то есть в направлении от процедуры P_n к процедуре P_1.

Отметим, что последовательности P_1^n соответствует путь в графе вызовов программы P. При этом порядок обхода процедур, определенный на первом шаге, гарантирует, что перед рассмотрением некоторой процедуры, все процедуры, вызываемые из нее, уже рассмотрены. При таком порядке обхода для любой последовательности P_1^n процедуры будут рассмотрены в направлении от процедуры P_n к процедуре P_1.

Решение задачи анализа приватных переменных для некоторой процедуры начинается с анализа внутренней области данной процедуры. Разработанный алгоритм основан на анализе потока данных (data-flowanalysis)[12]. Каждый цикл процедуры рассматривается отдельно. Анализ состоит из трех частей:

1. Анализ достигающих определений [12] определяет, в каких базовых блоках графа управления анализируемой процедуры может быть определена каждая переменная процедуры при достижении потоком управления каждого базового блока.
2. Анализ внутренней области цикла выполняется для проверки условия (1) из определения приватной переменной.
3. Анализ операций, внешних по отношению к циклу, выполняется для проверки условия (2) из определения приватных переменных.

При анализе внутренней области цикла анализируется подграф графа управления, вершинами которого являются базовые блоки, входящие в анализируемый цикл. Дуги графа управления, являющиеся для цикла обратными, входными и выходными, отбрасываются. Для каждого базового блока, входящего в подграф, определяется состояние переменных используемых процедурой. Для этого выполняется анализ потока данных в данном подграфе.Анализ выполняется в прямом направлении. Входным базовым блоком является заголовок цикла.

Множество значений потока данных описывает состояния переменных в некотором базовом блоке:

- Неопределенное состояние (UNKNOWN), означает, что анализ базового блока еще не проводился.
- Переменная не использовалась в базовом блоке, и на любом пути, ведущем к данному блоку, переменная либо не использовалась, либо была проинициализирована (NOT_USE).
- В базовом блоке или на некотором пути в графе управления, ведущем к данному блоку, переменная была использована,не будучи проинициализированной (NOT_INIT).
- Переменная была использована в базовом блоке на чтение, не будучи проинициализированной, после чего она была проинициализирована в том же базовом блоке (INIT_AFTER_USE).
- На выходе из базового блока переменная является проинициализированной (INIT).

Оператор сбора задается с помощью **Таблицы 1**.В первой строке и столбце таблицы указаны состояния переменных в базовых блоках, к которым применяется оператор сбора. Оператор сбора идемпотентен, коммутативен и ассоциативен. Верхним элементом полурешетки, образованной множеством значений потока данных и оператором сбора, является UNKNOWN, нижним –NOT_INIT.

Таблица 1. Оператор сбора для двух базовых блоков

	UNKNOWN	NOT_USE	NOT_INIT	INIT_AFTER_USE	INIT
UNKNOWN	UNKNOWN	NOT_USE	NOT_INIT	INIT	INIT
NOT_USE	NOT_USE	NOT_USE	NOT_INIT	NOT_USE	NOT_USE
NOT_INIT	NOT_INIT	NOT_INIT	NOT_INIT	NOT_INIT	NOT_INIT
INIT_AFTER_USE	INIT	NOT_USE	NOT_INIT	INIT_AFTER_USE	INIT
INIT	INIT	NOT_USE	NOT_INIT	INIT	INIT

Передаточнаяфункциязадается с помощью **Таблицы 2**. В первой строке таблицы показаны значения после применения оператора сбора к базовым блокам, предшествующим исследуемому базовому блоку. В первом столбце показаны значения после локального анализа исследуемого базового блока. Локальный анализ выполняется в предположении того, что используемые в базовом блоке переменные не проинициализированы и определяет локальное состояние переменных согласно следующим правилам:

- NOT_INIT – первый использующий переменную оператор использует ее на чтение и нет ни одного оператора, использующего данную переменную на запись;
- INIT_AFTER_USE – первый использующий переменную оператор, использует ее на чтение, но существуетоператор, присваивающий переменной некоторое значение;
- INIT – первый использующий переменную оператор использует ее на запись;
- NOT_USE – переменная не используется в базовом блоке.

Таблица 2. Передаточная функция для базового блока

	UNKNOWN	NOT_USE	NOT_INIT	INIT_AFTER_USE	INIT
NOT_USE	UNKNOWN	NOT_USE	NOT_INIT	INIT	INIT
NOT_INIT	UNKNOWN	NOT_INIT	NOT_INIT	INIT	INIT
INTI_AFTER_USE	UNKNOWN	INIT_AFTER_USE	INIT_AFTER_USE	INIT	INIT
INIT	INIT	INIT	INIT	INIT	INIT

Начальная инициализация состояний переменных в заголовке цикла, являющемся входным базовым блоком при анализе подграфа, выполняется следующим образом:

- Состояние INIT устанавливается для индуктивных переменных цикла.
- Для остальных переменных устанавливается состояние NOT_USE.

Для остальных базовых блоков, входящих в подграф, согласно алгоритму анализа потока данных состояние переменных устанавливается равным верхнему элементу (UNKNOWN)полурешетки, образованной множеством значений потока данных и оператором сбора.

При анализе используется информация,содержащаяся в графе памяти, о пересечении между участками памяти, соответствующими разным переменным.

На основе информации о состоянии переменных, полученной в результате анализа потока данных в заданном подграфе, определяется множество переменных, являющихся кандидатами в приватные переменные, множество переменных используемых в цикле только на чтение (при распараллеливаниитакие переменные могут быть объявлены как общие), множество неиспользуемых переменных в цикле.При определении данных множеств анализатор руководствуется следующими правилами:

- Согласно условию (1)из определения приватной переменной на каждой итерации анализируемого цикла приватная переменная перед использованием должна быть проинициализирована некоторым значением. Это означает, что для каждого базового блока, входящего в цикл, приватная переменная не может находиться в состоянии NOT_INITили INIT_AFTER_USE.
- Если для всех базовых блоков, входящих в анализируемый цикл, переменная находится в состоянии NOT_USE, то данная переменная не используется в анализируемом цикле.
- Переменные, используемые в цикле только на чтение, ни в одном базовом блоке не могут находиться в состоянии INITили INIT_AFTER_USE.
- Все остальные переменные не являются ни приватными, ни приватными по выходу и вызывают в цикле зависимости по данным.

На основе кандидатов в приватные переменные определяется множество кандидатов в переменные приватные по выходу.

Данный анализ выполняется с использованием результатов анализа достигающих определений. Рассматриваются все базовые блокипроцедуры, не входящие ванализируемый цикл. Переменная, являющаяся кандидатом в приватные переменные, становится кандидатом в переменные приватные по выходу при выполнении следующих условий для некоторого базового блока, не входящего в цикл:

1. Для данной переменной перед входом в базовый блок среди достигающих определений есть определение, находящееся внутри цикла.
2. Первое использование данной переменной в базовом блоке является использованием на чтение, то есть локальный анализ базового блока показал, что переменная находится в состоянии NOT_INITили INIT_AFTER_USE.

Если для формальных параметров процедуры, элементов COMMON-блоков и модулей в конечном базовом блоке графа управления анализируемой процедуры доступны определения данных переменных,локализованные внутри анализируемого цикла, то данные переменные должны быть дополнительно проанализированы с использованием межпроцедурного анализа.В этом случаедля процедур, вызывающиханализируемую процедуру, проверяется выполнение условий аналогичных условиям, описанным выше. Базовые блоки, содержащие достигающие определения анализируемой переменной (условие 1), ищутся среди базовых блоков, содержащих вызовы анализируемой процедуры.

После того как были найдены кандидаты в переменные приватные по выходу, необходимо проверить выполнение следующих условий:

- Из анализируемого цикла существует только один выход. В противном случае нельзя гарантировать выполнение условия (2) для приватных по выходу переменных. Следовательно, все кандидаты в приватные по выходу переменные помечаются как переменные вызывающие зависимость.
- При любом выполнении анализируемого цикла (не зависимо от состояния памяти программы) на последней итерации цикла данная переменная всегда получает одно и то же значение. Нарушение данного условия возможно в случае, если в цикле несколько обратных дуг. В этом случае перед выходом могут выполняться разные базовые блоки (при этом выход из цикла только один), в которых анализируемая переменная будет определена по разному. Для такого анализа используется результат анализа достигающих определений.

Необходимый межпроцедурный анализ выполняется только в том случае, если данная проверка прошла успешно.

Для программы на **Рис. 2**проведенный анализ для цикла с заголовком в строке 7 показал:

- переменная `j`и член COMMON-блока`a`являются приватными по выходу;
- переменная `i`не является приватной и вызывает зависимость;
- именованная константа `n`только читается.

6. Программная реализация

Анализатор реализован на языке программированияC++ с использованием библиотек STL (StandardTemplateLibrary) и BGL(BoostGraphLibrary). BGLявляется одной из библиотек, входящих вBoost [16]. Кроме этого использовались и другие библиотеки, входящие в Boost.

Анализатор написан в объектно-ориентированном стиле с применением идей метапрограммирования. Одним из примеров является реализация схемы алгоритма анализа потока данных. Разработан шаблонный класс описывающий последовательность действий, выполняемую при анализе. Для использования данного класса достаточно определить передаточные функции для базовых блоков, оператор сбора и множество значений потока данных.

Программа анализатора состоит из нескольких частей, реализованных в виде библиотек:

- Библиотека, содержащая базовые структуры данных. В ней определены общие компоненты, не зависящие от анализатора и используемые остальными его частями.Например, здесь определено общее представление графов в анализаторе, построенное на основе BGL.
- Библиотека, содержащая структуры внутреннего представления анализатора.
- Библиотека, содержащая реализацию ввода/вывода. Данная библиотека предоставляет средства для доступа к базе данных САПФОР. На их основе в ней реализованfrontendанализатора, кроме того средства ввода/вывода используются для сохранения результатов анализа.
- Библиотека, содержащая модули анализа, является ядром анализа.

Все компоненты программы анализатора сопровождаются комментариями в формате, поддерживаемом средством автоматической генерации документации Doxygen [17]. Программа анализатора насчитывает 12000строк кода.

7. Экспериментальная проверка

Анализатор был проверен экспериментально на тесте Якоби, программах Каверна и Контейнер (ИПМ им. М.В. Келдыша РАН)и др.

Корректность результатов анализа для теста Якоби была проверена вручную с помощью диалоговой оболочки САПФОР, а также с помощью эксперта САПФОР. Для нее была получена корректно работающая параллельная версия программы.

Для остальных задач, в силу их объема, проверка корректности возможна только с использованием экспертов САПФОР. На основе результатов анализа, сохраненных в базе данных, были построены параллельные версии для программ Контейнер и Каверна.

Время работы анализатора на персональном компьютере с процессором Intel® Core™2DuoT8300c частотой 2,4 Ггц и 3 Гб оперативной памяти приведено в **Таблице 3**.

Таблица 3. Время работы анализатора

Анализируемая программа	Количество строк	Время, затраченное на анализ (сек.)
Якоби	40	< 1
Каверна	500	2
Контейнер	900	6
Приложение 1	3000	16
Приложение 2	19000	977 (16 минут)

Из детализации данных замеров времени,доступной в процессе анализа, видно, что основное время затрачено на анализ достигающих определений и поиск приватных переменных. Из этого следует, что деятельность по оптимизации работы анализатора должна быть сконцентрирована на этих двух стадиях анализа.

8. Заключение

Распараллеливание последовательных программ требует четкого понимания информационной структуры разрабатываемой программы. Для этого в рамках САПФОР было решено разработать статический анализатор. Разработка анализатора выполняется в три этапа:

1. Создание подсистемы, решающей задачу анализа приватных скалярных переменных.
2. Интеграция существующего анализатора в разрабатываемый анализатор с целью определения информационных и регулярных зависимостей.
3. Создание подсистемы, выполняющейанализ приватных массивов.

Первый этап разработки выполнен. В данный момент осуществляется реализация второго этапа. Выполнение третьего этапа запланировано на 2012 год.

В рамках первого этапа было разработано внутреннее представление, независящее от языка программирования, на котором написана анализируемая программа и решена задача анализа приватных скалярных переменных. Написанfrontendдля базы данных САПФОР, построенной на основе языка программирования Fortran95.

Анализатор проверен на программах, содержащихдо 20 тысяч строк. Анализ был выполнен за приемлемое время. Его корректность была проверена с помощью диалоговой оболочки и экспертов САПФОР.

Литература

1. TOP500 List– November 2011. URL:http://www.top500.org/list/2011/11/100 (датаобращения 14.01.2012)

2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2004. – 608 с.
3. OptimizingTechnologies.URL: http://www.optimitech.com/ (датаобращения: 15.01.2012).
4. ParallelSoftwareProductsInc.URL: http://www.parallelsp.com/ (датаобращения: 14.01.2012).
5. Юрушкин М.В., Петренко В.В., Штейнберг Б.Я., Алымова Е.В.,Абрамов А.А., Баглий А.П., Гуда С.А., Дубров Д.В., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Полуян С.В., Скиба И.С., Шаповалов В.Н., Штейнберг О.Б., Штейнберг Р.Б. Диалоговый высокоуровневый оптимизирующий распараллеливатель (ДВОР) // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи” (20-25 сентября 2010 г., г. Новороссийск). - М.: Изд-во МГУ, с. 71-75.
6. Система АвтоматизированнойПараллелизации Фортран Программ.URL: http://www.keldysh.ru/dvm/SAPFOR/ (дата обращения: 02.01.2012).
7. Бахтин В.А., Коновалов Н.А., Крюков В.А., Поддерюгина Н.В. FortranOpenMP/DVM – язык параллельного программирования для кластеров// Материалы второго Международного научно-практического семинара “Высокопроизводительные параллельные вычисления на кластерных системах”, г. Нижний Новгород, 26-29 ноября 2002 г., с.28-30.
8. Бахтин В.А., Коновалов Н.А., Поддерюгина Н.В., Устюгов С.Д. Гибридный способ программирования DVM/OpenMP на SMP-кластерах // Труды Всероссийской научной конференции “Научный сервис в сети Интернет: технологии параллельного программирования” (сентябрь 2006 г., г. Новороссийск), Изд-во Московского Университета, с.128-130.
9. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: экзафлопсное будущее” (19-24 сентября 2011 г., г. Новороссийск). - М.: Изд-во МГУ, с. 310-315.
10. Крюков В.А., Клинов М.С., Бахтин В.А., Поддерюгина Н.В. Автоматическое распараллеливание последовательных программ для многоядерных кластеров// Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи” (20-25 сентября 2010 г., г. Новороссийск). - М.: Изд-во МГУ, с. 12-15.
11. Бахтин В. А., Бородич И.Г., Катаев Н.А., Клинов М.С., Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР. // Труды Международной научной конференции “Научный сервис в сети Интернет: экзафлопсноебудущее” (19-24 сентября 2011 г., г. Новороссийск). – М.: Изд-во МГУ, 2011, с. 67-70
12. Ахо А.В., Лам М.С., Сети Р., УльманДж.Д. Компиляторы: принципы, технологии и инструментарий, 2-е издание: Пер. с англ.- М.ООО "И.Д.Вильямс", 2008.-1184 с.Главы 9, 11.
13. pC++/Sage++ Home Page. URL: http://www.extreme.indiana.edu/sage/ (датаобращения 12.01.2012)
14. Катаев Н.А. Система автоматизированного распараллеливания Фортран-программ: анализ многомодульных программ // Сборник тезисов лучших дипломных работ 2009 года. – М.: Издательский отдел Факультета ВМиК МГУ им. М.В. Ломоносова; МАКС Пресс, 2009. С. 141 – 142.
15. Катаев Н.А. Анализ последовательных программ с помощью средств УБТ // Труды международной научной конференции “Параллельные вычислительные технологии (ПаВТ’2011)” (Москва, 28 марта – 1 апреля 2011 г.) – Челябинск: Издательский центр ЮУрГУ, 2011, с. 697.
16. Boost C++ Libraries.URL: http://www.boost.org/ (датаобращения 12.01.2012)
17. Doxygen. URL: http://www.doxygen.org(дата обращения 12.01.2012)